

Forum:[Hack Forums](#)

Thread: [Using TDS to bypass detection tutorial](#)

littlephish420 - September 18, 2023, 6:37 pm

"If you do not have a TDS API, you can join the free beta program here: <https://hackforums.net/showthread.php?tid=6251680> and get a TDS API"

Introduction

Today we will explore how to harness the power of a Traffic Distribution System (TDS) using a custom reverse proxy implemented in the Go programming language (Golang). A Traffic Distribution System (TDS) is a sophisticated tool designed to efficiently manage incoming network traffic. It can be used in both business scenarios, where it aids in preventing fraudulent activities, and in the realm of penetration testing, where it plays a crucial role in remaining undetected during legitimate engagements.

Understanding the Significance of TDS

TDS has gained widespread adoption in the business world due to its ability to mitigate risks such as fraud. Additionally, some TDS solutions even assist in detecting temporary email addresses and phone numbers. This versatility makes TDS an indispensable asset for organizations worldwide, helping them safeguard their operations and reputations.

Practical Application in Penetration Testing

In this article, we will dive into the practical application of the reverse proxy linked here within the context of a theoretical penetration testing engagement. We will employ a reverse proxy as an intermediary, intercepting requests from clients and efficiently routing them to the appropriate backend servers. By configuring this setup, we can ensure that our website remains invisible to requests from web crawlers, bots, antivirus systems (AVs), endpoint detection and response solutions (EDRs), and mechanisms like SafeBrowsing. This approach not only shields your backend but also safeguards it against being labeled as suspicious as known as "anti-red" in certain circles.

By the end of this article, you'll know how to leverage a TDS using a custom Golang reverse proxy, enhancing your ability to manage and protect your backend from malicious network traffic effectively. Let's dive into it...

Compiling the Reverse Proxy Project and Setting Up a TDS-Enabled Environment

In this section, we'll walk you through the process of compiling the reverse proxy project and creating an environment with Traffic Distribution System (TDS) capabilities. We'll be using a Debian Virtual Private Server (VPS) for this purpose and leverage the Go programming language to compile the reverse proxy binary.

1. Installing Git and Go

Begin by ensuring that Git is installed on your server. You can check everything working by running git in your console. If Git is not installed, install it using the following command:

Code `sudo apt install git` Once Git is ready, clone the Go installer repository from GitHub:

Code `git clone https://github.com/kerolloz/go-installer.git`

Navigate to the cloned directory:

Code `cd go-installer`

Execute the installer script to set up Go:

Code `./go.sh`

2. Creating the Reverse Proxy Directory and Configuration

With Go installed, it's time to create a directory for your reverse proxy project. Navigate to your desired location and create the "proxy" directory:

Code `mkdir proxy`

`cd proxy`

Inside the "proxy" directory, you'll want to create a file named "proxy.go" to host your reverse proxy code:

Code `touch proxy.go`

Use your preferred text editor to edit "proxy.go" (we'll use nano in this example):

Code `nano proxy.go`

Paste your reverse proxy code into this file and make any necessary adjustments, such as configuring the endpoint to point to the backend hosting your web files and specifying the TDS API. Save the changes.

3. Setting Up the Go Project

Initialize a Go module for your project:

Code `go mod init proxy`

Ensure that all required packages are installed:

Code `go mod tidy`

4. Compiling the Reverse Proxy

Compile your reverse proxy by running the following command:

Code `go build`

The following command will generate an ELF binary that you can run. Execute the binary with:

Code `./proxy`

Your reverse proxy will now be up and running, listening on port 80.

5. Configuring Your Backend Server

On your backend server, add security by configuring your firewall to only permit incoming traffic from your reverse proxy. This step ensures that your website's files are accessible to the reverse proxy so it can serve it so that your backend will not get detected as suspicious.

6. Pointing Your Domain to Your Debian VPS

Once your domain is ready, configure A records to point to your Debian VPS's IP address. After the domain propagates, share it with your intended audience. You'll notice that only legitimate visitors are granted access to your website files, thanks to the Traffic Distribution System (TDS) capabilities of your reverse proxy.

By following these steps, you'll have successfully set up a TDS-enabled environment using a

custom Golang reverse proxy, bolstering your website's security and performance.

This prevents detection as suspicious during legitimate pentesting engagements.

Proxy.go source code

Code package main

```
import (  
    "database/sql"  
    "fmt"  
    "net/http"  
    "net/http/httputil"  
    "net/url"  
    "io/ioutil"  
    "strings"  
    "time"  
    "crypto/md5"  
    "encoding/hex"  
    _ "github.com/mattn/go-sqlite3"  
    "github.com/fatih/color"  
)  
  
func main() {  
    db, err := sql.Open("sqlite3", "data.db")  
  
    if err != nil {  
        panic(err)  
    }  
  
    defer db.Close()  
  
    // Create the replay table if it doesn't exist  
    _, err = db.Exec(`CREATE TABLE IF NOT EXISTS replay (  
        id INTEGER PRIMARY KEY AUTOINCREMENT,  
        ip TEXT NOT NULL,  
        md5hash TEXT NOT NULL  
    );`)  
  
    if err != nil {  
        panic(err)  
    }  
  
    endpoint := "https://example.com"  
  
    target, err := url.Parse(endpoint)  
  
    if err != nil {  
        panic(err)  
    }  
  
    proxy := httputil.NewSingleHostReverseProxy(target)  
  
    // Start Proxy
```

```

server := http.Server{
    Addr: ":80", // Default port
    Handler: http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {

        ip := extractIPAddress(r)
        userAgent := r.Header.Get("User-Agent")

        // Check IP against SQL database to see if match exist to prevent replay attacks,
        https://en.wikipedia.org/wiki/Replay_attack
        md5hash := md5Hash(ip)
        if !isIPInDatabase(db, md5hash) {
            apiResponse, err := callAPI(ip, userAgent)
            if err != nil {
                fmt.Println("API request failed:", err)
                http.Error(w, "500", http.StatusInternalServerError)
                return
            } else {
                /* Debug Response
                fmt.Println("(DEBUG) Status Code:", apiResponse.StatusCode)

                responseBytes, err := ioutil.ReadAll(apiResponse.Body)

                if err != nil {
                    fmt.Println("(DEBUG) Error reading the response body of API:", err)
                }

                fmt.Println("(DEBUG) Response Body:", string(responseBytes))
                apiResponse.Body.Close() */
            }

            if apiResponse.StatusCode == http.StatusForbidden {
                color.Red(fmt.Sprintf("%s - IP: %s - MD5: %s - Connection Dropped (403)\n",
time.Now().Format(time.RFC3339), ip, md5hash))
                dropConnection(w)
                return
            } else if apiResponse.StatusCode == http.StatusOK {
                color.Green(fmt.Sprintf("%s - IP: %s - MD5: %s - Connection Allowed (200)\n",
time.Now().Format(time.RFC3339), ip, md5hash))
                proxy.ServeHTTP(w, r)

                /*
                * Uncomment this if you want people to not be able to replay a request...
                err := addToDatabase(db, md5hash)
                if err != nil {
                    fmt.Println("Error adding IP to database:", err)
                }
                return */
            }
        } else {
            // Replay attack detected, connection dropped.
            color.Red(fmt.Sprintf("%s - IP: %s - MD5: %s - Connection Dropped (Replay)\n",
time.Now().Format(time.RFC3339), ip, md5hash))

```

```

        dropConnection(w)
        return
    }
}),
}

// SSL should be implemented at some point...
fmt.Println("Server is started and runs on port 80")
if err := server.ListenAndServe(); err != nil {
    panic(err)
}
}

func md5Hash(text string) string {
    hash := md5.Sum([]byte(text))
    return hex.EncodeToString(hash[:])
}

func isIPInDatabase(db *sql.DB, md5hash string) bool {
    var count int
    err := db.QueryRow("SELECT COUNT(*) FROM replay WHERE md5hash = ?",
md5hash).Scan(&count)
    if err != nil {
        fmt.Println("Error checking IP in database:", err)
        return false
    }
    return count > 0
}

func addToDatabase(db *sql.DB, md5hash string) error {
    _, err := db.Exec("INSERT INTO replay (ip, md5hash) VALUES (?, ?)", "", md5hash)
    return err
}

func extractIPAddress(r *http.Request) string {
    ip := strings.Split(r.RemoteAddr, ":")[0]
    return ip
}

func callAPI(ip, userAgent string) (*http.Response, error) {
    api := fmt.Sprintf("http://beta.somedomain.placeholder/lookup.php?ip=%s&ua=%s", ip,
userAgent)

    resp, err := http.Get(api)
    if err != nil {
        return nil, err
    }

    return resp, nil
}

func dropConnection(w http.ResponseWriter) {
    // hijack inbound connection to make server seem offline

```

```
if hj, ok := w.(http.Hijacker); ok {  
    clientConn, _, err := hj.Hijack()  
    if err != nil {  
        fmt.Println("dropping connection error:", err)  
        return  
    }  
    clientConn.Close()  
}  
}
```